(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 265 152 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
27.07.2005 Bulletin 2005/30

(51) Int Cl.[7]: **G06F 17/30**, G06F 12/02, G06F 3/06

(21) Application number: **01127837.1**

(22) Date of filing: **22.11.2001**

(54) **Virtual file system for dynamically-generated web pages**

Virtuelles Dateisystem für dynamisch erzeugte Webseiten

Système de fichiers pour des pages web générés de manière dynamique

(84) Designated Contracting States:
DE FR GB NL

(30) Priority: **04.06.2001 US 873878**

(43) Date of publication of application:
11.12.2002 Bulletin 2002/50

(73) Proprietor: **Hewlett-Packard Company**
Palo Alto, CA 94304 (US)

(72) Inventor: **Holcomb, David Marshall**
San Diego, CA 92127 (US)

(74) Representative: **Liesegang, Eva et al**
Forrester & Boehmert,
Pettenkoferstrasse 20-22
80336 München (DE)

(56) References cited:
EP-A- 1 041 572
WO-A-00/57315
WO-A-99/42934
US-A- 4 775 969
US-A- 5 991 753
US-A- 6 081 883

- **MAKAROFF D J ET AL: "Disk cache performance for distributed systems" PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS. PARIS, MAY 28 - JUNE 1, 1990, LOS ALAMITOS, IEEE COMP. SOC. PRESS, US, vol. CONF. 10, 28 May 1990 (1990-05-28), pages 212-219, XP010019312 ISBN: 0-8186-2048-X**

EP 1 265 152 B1

Printed by Jouve, 75001 PARIS (FR)

Description

Cross-Reference to Related Applications

[0001] This application relates to the subject matter disclosed in the co-pending U.S. application Ser. No. 09/686,553 (attorney docket 10003752-1), by Holcomb et al., filed October 11, 2000, titled "Dynamically-Generated Web Pages". This application is assigned to the assignee of the present invention.

Field of the Invention

[0002] The present invention relates generally to file systems for mass storage devices, and pertains more particularly to a virtual file system for a peripheral device that provides a large block-oriented mass storage device interface to a host system while using a much smaller amount of memory in the peripheral device and dynamically generating the data blocks read by the host.

Background of the Invention

[0003] Mass storage peripherals, such as disk drives, are in common use in computer systems today for storing and retrieving data. This data is stored in blocks. Many different logical sets of data, commonly referred to as "files", can be stored on the peripheral. While the data blocks are typically of a fixed size, a file can contain more data than can fit in a single data block, and so a file may consist of a number of data blocks. In addition, the various data blocks that make up a file do not need to be located together, but rather can be stored in different places on the peripheral. In order to facilitate access to files of data by the computer system, mass storage peripherals typically provide a file system which describes the organization on the peripheral of the data blocks for each file.

[0004] With conventional mass storage peripherals, data blocks are read and written only under the explicit control of the computer system. In other words, the computer system can keep its own independent record of where certain data blocks and files are stored on the peripheral and be assured that, in fact, that is where the data blocks and files are located. In order to speed up the storage and retrieval of frequently used data blocks, computer systems also frequently implement a data block cache internal to the computer system itself that stores these frequently accessed data blocks. The computer system can read a data block from the cache much more quickly than it can from the peripheral, and so computer systems typically search their cache for a desired data block, and request it from the peripheral only if it is not found in the cache.

[0005] The above-described caching mechanism works well in the case of traditional mass storage peripherals, such as disk drives, in which data blocks are only read and written under the explicit control of the computer system. Such disk drives also generally have large enough amounts of memory to store the tables that implement the file system, as well as the data blocks for all the files. However, certain intelligent mass storage peripherals, such as one described in the above-referenced co-pending and commonly owned patent application by Holcomb et al., may dynamically create, delete, or rearrange data blocks independent of the explicit control of the computer system. As a result, the computer system can no longer be assured that the data blocks in its cache match the data blocks on the peripheral, and incorrect operation can result when the computer system reads cached files rather than obtaining files from the peripheral. In order to ensure proper operation, the peripheral must inform the computer system each time it dynamically creates, deletes, or rearranges data blocks, and the computer system must delete and rebuild the cache and other file system tables. Some computer systems provide a mechanism for the peripheral to so inform the computer system; however, the actions to delete and rebuild the cache and other file system tables can be time-consuming and inefficient. Other computer systems do not support such a mechanism at all.

[0006] A related limitation of non-traditional peripherals is that they may not contain sufficient memory to store all the data blocks needed to provide a large file system. As a result, the number and size of files stored on the peripheral may be disadvantageously limited, thus requiring the peripheral to dynamically delete certain files and create others, and giving rise to the previously discussed synchronization problem between the computer system and the peripheral.

[0007] The following prior art documents may be of interest with respect to the present invention. US-A-6 081 883 discloses a scalable computer system comprising an interconnect bus, a host-processor coupled to the interconnect bus and one or more function-specific processors including a network processor, a file processor and a storage processor, the one or more function-specific processors coupled to the interconnect bus for communicating with other processors, each function-specific processor having a buffer-memory coupled thereto, the buffer memory having one or more segments which serve as function-specific caches to cache function-specific data including network-processing data, file-processing data and storage processing data, the segments being dynamically allocable to different processors. Furthermore, the computer system may comprise a file storage processor (FSP) coupled to the interconnect bus, one or more data storage devices coupled to the FSP, each data storage device having files and associated meta data; a meta data cache coupled to the FSP for buffering the meta data; and a write cache coupled to the FSP for buffering file write operations from the FSP.

WO 00/57315 A discloses a system for maintaining file object data, comprising a server component, the server

component associated with a remote storage mechanism that maintains at least some of the file object data, and a client component, the client component associated with the local storage mechanism that maintains at least some of the file object data, the client component configured to receive a request for a file object data and to determine whether the file object data is locally maintained or remotely maintained, and when locally maintained, to provide access to the locally maintained file data, and when maintained remotely to communicate with the server component to provide access to the remotely maintained file data.

[0008] EP-A-1 041 572 discloses a non-volatile memory and a non-volatile memory reproducing apparatus for segmenting a signal data file into blocks each having a predetermined length and adding an attribute file for managing the single data file to each single data file, the non-volatile memory having a data area for reproduction management file for managing a plurality of files each of which is composed of the blocks; and a file management area for file management information for managing the plurality of data files and the reproduction management file. According to one embodiment, the file management information is a file allocation table file and the attribute file added to each single data file contains a number of blocks that compose each single data file and the reproduction management file contains data representing the reproduction order of a plurality of data files recorded in the data area.

[0009] It is one object of the present invention to provide a new and improved file system and method of accessing a large number of data files which allows files to be created, deleted, and rearranged without requiring the host computer system to delete and rebuild its cache and other file system tables and without requiring a large amount of memory.

[0010] This object is achieved by a method according to claim 1 and by an apparatus according to claim 9.

Summary of the Invention

[0011] In a preferred embodiment, the present invention provides a virtual filesystem capable of including a large number of data files. The virtual filesystem has a predetermined structure of directories and files, and a linear filespace. The linear filespace includes a directory region and a file allocation table region in which, as a result of the predetermined nature of the directory structure, the data blocks are calculatable upon a read request and are not required to be stored in the filesystem. The linear filespace also includes a file region having a predefined number of predefined-size files. Each of a set of lowest-level directories of the directory structure contains a single file. The particular single file in the file region that is linked to a particular lowest-level directory is preferably determined at the time when the lowest-level directory is first accessed, thus allowing a previously-unused or least-recently-used one of the files to be allocated so as to avoid the need to delete and rebuild the host's cache. The data blocks of the file are preferably dynamically generated and provided to the virtual filesystem upon request, or aliased to a file stored in an auxiliary filesystem that is read via the virtual filesystem.

Brief Description of the Drawings

[0012] The above-mentioned features of the present invention and the manner of attaining them, and the invention itself, will be best understood by reference to the following detailed description of the preferred embodiment of the invention, taken in conjunction with the accompanying drawings, wherein:

FIG. 1 is a schematic representations of an exemplary predefined file structure for a virtual filesystem according to the present invention;
FIG. 2 is a schematic representations of the filespace of the virtual filesystem of FIG. 1;
FIG. 3 is a more detailed schematic representations of an exemplary directory region of the filespace of FIG. 2;
FIG. 4 is a more detailed schematic representations of an exemplary file allocation table region of the filespace of FIG. 2;
FIG. 5 is a flowchart of a method for reading data from the virtual file system of FIG. 1;
FIG. 6 is a more detailed flowchart of a data block calculation portion of the flowchart of FIG. 5; and
FIG. 7 is a block diagram of an apparatus for providing the virtual file system of FIG. 1.

Description of the Preferred Embodiment

[0013] Referring now to the drawings, there is illustrated a virtual filesystem 10 (also referred to as a "VFS") constructed in accordance with a novel method 100 of reading data from an apparatus 50 on which the filesystem 10 is implemented. The virtual filesystem 10 advantageously provides a large virtual filespace that can be implemented in only a relatively small amount of memory by predefining the directory hierarchy such that data blocks corresponding to the directories (and an associated file allocation table, also referred to herein as a "FAT") can be algorithmically calculated, and by binding data files to the directory hierarchy only at or near the time of use such that data blocks corresponding to the data files can be dynamically generated at or near the time of access. The present invention is particularly applicable to FAT-based file systems such as the widely-used MS-DOS file system that is well known to those skilled in the art.

[0014] As is understood by those skilled in the art, a filesystem is a structure of directories, files, and other internal information which an operating system uses to locate and access these directories and files. The directories are typically organized in a directory hierarchy in

which some directories are related to other directories in a parent-and-child tree structure. Files are located within directories. The data structures that comprise the filesystem occupy a preferably linear filespace in memory. A virtual filesystem is one in which the directories and files appear to exist in a linear filespace when viewed by a host computer or other device which is external to the filesystem, but in actuality may not exist in linear filespace or may even not exist in physical memory at all until the host computer requests the filesystem to perform an operation on a directory or file.

[0015] As is best understood with reference to the exemplary predefined file structure 15 of FIG. 1, the virtual filesystem 10 of the present invention has a predefined and fixed directory hierarchy 20, which begins at a root directory "\" 21. Below the root directory 21 is a set 22 of 1st-level directories which are subdirectories to the root directory. The exemplary directory hierarchy 20 contains four 1st-level directories, denoted D1 through D4. The number of subdirectories contained below a higher-level directory is referred to as the "fanout"; the exemplary directory structure uses a fanout of four at all levels, but for clarity only one set of four subdirectories is illustrated in FIG. 1 for each level. 1st-level directory D2 23 contains four 2nd-level directories, which are denoted D9 through D 12. 2nd-level directory D11 24 contains four 3rd-level directories, which are denoted D45 through D48. Both the 1st-level directories and the 2nd-level directories are considered to be intermediate-level directories in that they have both a higher-level ("parent") directory and lower-level ("child") directories. The Yth-level directories (in this exemplary hierarchy 20 the 3rd-level directories) are the lowest-level directories in the directory hierarchy 20. Each Yth-level directory according to the present invention contains one of a set 18 of predefined file placeholders, such as directory D48 25 contains the file placeholder F88 26; the lowest-level directories have a parent directory but no child directories. The significance of the directory and file labeling used in FIG. 1 will become apparent when FIG. 3 is discussed subsequently.

[0016] The directory fanout N and hierarchy depth Y can be advantageously chosen to provide a virtual filesystem having a desired number of files. The number of files is calculated according to the formula N^Y. The exemplary hierarchy 20 with a fanout of 4 and a depth of 3 thus provides $4^3 = 64$ files. The preferred embodiment has a fanout of 32 and a depth of 4, and thus provides $32^4 = 1,048,576$ files. While the fanout N and the depth Y may be different at each level and branch of the directory hierarchy, in the preferred embodiment they are the same at all levels and branches, as will become apparent when the method of the present invention is discussed subsequently.

[0017] As best understood with reference to FIG. 2, the virtual file system occupies a filespace 30 which contains B logical data blocks, such as data block 2. Each data block contains b bytes. In the commonly-used FAT-32 file system well known to those skilled in the art, each block typically contains 4k bytes of data, a data block can be further divided, typically according to the type of physical media on which the filesystem is implemented, but such discussions will be omitted here since they are also well known in the art and are not germane to the present invention.

[0018] The data blocks of the virtual filesystem 10 are preferably organized sequentially. The data blocks can be requested, typically by block number, from the virtual file system 10 by a host computer (not shown) or other device. The filespace 30 is logically divided into several contiguous regions. A directory region 32 of D blocks in size predefines the directory hierarchy of the virtual filesystem 10, as explained heretofore.

[0019] The filespace 30 also contains a number of file regions 34a,b,c which in total occupy F blocks of the filespace 30. Each file region 34a,b,c predefines a quantity of data files of a predetermined number of data blocks in size. File region 34a defines a quantity Q_1 of data files, each of which is one block in size. File region 34b defines a quantity Q_2 of data files, each of which is two blocks in size. File region 34c defines a quantity Q_M of data files, each of which is M blocks in size. The size of each file region is determined by the quantity of files and the block size of each file. The number of file regions, the quantity of files in each region, and the block size of the files in each region are not limited by the present invention but rather are design choices available to the filesystem 10 designer.

[0020] The filespace 30 also contains one or more FAT regions 36. Each FAT region 36 predefines a file allocation table of T blocks in size for locating in the filespace 30 all the data blocks for the directories and the data files. According to the present invention, and as will discussed subsequently in greater detail, these regions are organized so as to allow the data blocks in the directory region 32 and the FAT region 36 to be algorithmically calculated when the filesystem 10 is read in order to reduce the amount of physical memory required to contain the filesystem 10.

[0021] Considering now in further detail, and with reference to FIG. 3, the directory region 32, the directory region 32 contains directory entries. Each directory entry contains different fields which identify (at a minimum) the name, the starting block number, the attributes of the entry (in particular, whether it represents a child directory or a file), and the size of each file or child directory. Directory entries describe the contents of each individual directory of the directory hierarchy 20. In the preferred embodiment, the directory entries for each directory of the hierarchy 20 are contained in a separate data block whose position in the filespace 30 is calculable from the directory hierarchy 20. To illustrate using the exemplary directory hierarchy 20 of FIG. 1, the directory entries for the root directory 21 are assigned to block 0 of directory region 32. Block 0 contains four directory entries, which correspond to the directory entries

for the four 1st-level directories (D through D4) of the hierarchy 20. The numbers shown within each block of the directory region 32 of FIG. represent the value of the starting block field of the particular directory entry in the block, and correspond to the directory numbers used in the directory hierarchy 20. For example, the second entry 38a of block 0 corresponds to 1st-level directory D2, and indicates that the directory entries for directory D2 may be found in block 2. The third entry 38b of block 2 corresponds to 2nd-level directory D11, and indicates that directory entries for directory D 11 may be found in block 11. The fourth entry 38c of block 11 corresponds to Yth-level directory D48, and indicates that directory entries for directory D48 may be found in block 48. The first (and only) entry 38d of block 48 corresponds to a file placeholder. The file placeholder may or may not have an actual file associated with it at any particular point in time. The value of the starting block field for the file entry indicates the state of the file's existence. One predefined value indicates that no file has ever been created in directory D48, while a different predefined value indicates that a file was previously created in directory D48 but the file is no longer in existence. If the starting block field contains a value between 0 and F-1, then it indicates that an associated file currently exists, and the starting block field value is the block number in the file region 34 at which the file data is located. As will be described subsequently in further detail, a file placeholder for a Yth-level directory is bound to a particular data file only when a particular Yth-level directory is first accessed.

[0022] The above-described manner of storing N-ary (for example, binary, trinary, etc.) trees such as the directory hierarchy 20 in an array is well known to those skilled in the art. Because the directory hierarchy 20 is fixed and the positions in the filespace 30 of the data blocks for any particular directory are calculable, as will be described subsequently in further detail, a VFS 10 according to the present invention does not need to store the data for the directory region 32, but rather can algorithmically calculate the data for any particular data block when requested by the host computer.

[0023] Considering now in further detail the FAT region 36, and with reference to FIG. 4, in many filesystems the data blocks of a file or directory which spans more than one block are not located contiguously in the filespace 30. While a directory entry, as has been explained previously, indicates the starting block at which data for a directory or file will be found, it does not indicate the location of any additional data blocks. The FAT table provides this linkage between the starting block and any additional blocks. Because the directory hierarchy, and the number and size of files, are predetermined in the VFS 10 of the present invention, the contents of the FAT region are correspondingly calculable algorithmically.

[0024] The size of the FAT region 36 is determined by the number of data blocks contained in the VFS 10. The FAT 36 contains a FAT entry 42 for each data block in the system. Each FAT entry 42 must be of a size large enough to contain a value that represents the largest block number in the combined directory 32 and file 34 regions (i.e. the value of D+F). For example, the FAT-32 file system uses a four byte FAT entry 42, which is sufficient to address up to 2^{32} = 4,294,967,296 blocks.

[0025] The FAT region 36 includes a directory FAT subregion 36a and a file FAT subregion 36b. Since, as previously discussed, each directory occupies a single data block in the directory region 32, the number of FAT entries 42 in the directory FAT subregion 36a is equal to the number of directories in the directory hierarchy 20. For instance, in the exemplary directory hierarchy 20 of FIG. 1 in which N=4 and Y=3, there is one root directory, N^1=4 1st-level directories, N^2=16 2nd-level directories, and N^3=64 Yth (3rd)-level directories, for a total of 85 directories; therefore, there are 85 FAT entries 42 in the directory FAT subregion 36a.

[0026] With regard to the file FAT subregion 36b, different files can occupy a different number of data blocks in the file region 34. However, according to the filesystem 10 of the present invention, and as previously described, the number of files and the size of each file are predetermined, so the number of FAT entries 42 required for the FAT subregion 36b is algorithmically calculable. For instance, in the exemplary directory hierarchy 20 of FIG. 1 which contains $N^Y = 4^3 = 64$ files, assume that 32 (Q_1) of the files are 1-block files, 16 (Q_2) are 2-block files, 10 (Q_3) are 4-block files, and 6 (Q_4) are 8-block data files. Therefore, 32x1=32 blocks are required for the 1-block files, 16x2=32 blocks are required for the 2-block files, 10x4=40 blocks are required for the 4-block files, and 6x8=48 blocks are required for the M-block files (where M=8); for a total of 32+32+40+48 = 152 blocks. Therefore, the file FAT subregion 36b will contain 152 FAT entries 42.

[0027] The content of a FAT entry 42 for a particular data block in the directory region 32 or the file region 34 specifies whether the data for that directory or file continues in another data block and, if so, at which block the data continues. If the FAT entry 42 contains a value that corresponds to another block number in the filesystem 10, then the data continues at that block number. If the FAT entry 42 contains a predefined "end-of-chain" value that does not correspond to a block number in the filesystem 10, then the data does not continue in any additional block. With reference to the exemplary FAT table 36 of FIG. 4, the starting block of a 4-block data file is located at datafile-relative FAT entry #64 42'. In the exemplary file FAT subregion 36b of FIG. 4, FAT entry #64 42' contains the value 65, indicating that the 4-block data file continues at data block #65. Similarly, FAT entry #65 contains the value 66 indicating that the 4-block data file continues at data block #66, and FAT entry #66 contains the value 67 indicating that the 4-block data file continues at data block #67. Since data block #67 is the last of the four blocks in the chain, FAT

entry #67 contains the "end-of-chain" marker.

[0028] Considering now a novel method 100 of reading data from the virtual filesystem 10, and with reference to FIG. 5, the method 100 begins at 102 by defining a virtual file system 10 having a filespace 30 containing a logical linear arrangement of data blocks for a predetermined file structure 15 which includes a directory hierarchy 20 and a set of placeholders for predetermined file 18. At 104, a request to provide a data block 2 having a specified position in the VFS 10 is received, typically from the host computer connected to the VFS 10. At 106, the section of the VFS 10 in which the data block 2 is located is determined. If the data block 2 is located in the FAT region 36 or the directory region 32 ("FAT or Directory" branch of 106), then the method continues at 108.

[0029] If the data block 2 is in the FAT region 36, or if the data block 2 is in the directory region 32 but corresponds to the root directory or one of the intermediate-level directories in the directory hierarchy 20 ("No" branch of 108), then at 114 the content of the data block 2 is calculated based on the predetermined directory hierarchy 20 and the specified block position, and at 116 the content is provided to the host computer in response to its request. The algorithm by which the data block content is calculated will be described subsequently; calculation of the data block content is advantageously enabled by the novel predefined organization of the file structure 15 and filespace 30 of the VFS 10.

[0030] If the data block 2 is in the directory region 32 and corresponds to one of the lowest-level directories in the directory hierarchy 20 ("Yes" branch of 108), then at 110 the actual amount of data to be contained by the file corresponding to the lowest-level directory is determined. This is preferably done by a process external to the virtual filesystem 10 which has previously associated the file placeholder with the data, as will be described subsequently with reference to the virtual filesystem apparatus 50. Once this actual file size in bytes is known, at 112 the lowest-level directory is bound to an unused (or least recently used) predetermined file having a maximum file size which is greater than or equal to the amount of data, and the method continues at 114. Typically the smallest predetermined file of sufficient size to contain the data will be bound to the particular lowest-level directory.

[0031] By calculating data blocks 2 in the FAT region 36 and the directory region 32, the present invention advantageously minimizes the amount of physical memory that is required by the virtual filesystem 10, particularly as large numbers of files and large file sizes are included in the filesystem 10.

[0032] If the data block 2 is located in the file region 34 ("File" branch of 106), then at 116 the data source for the file is determined. If the data is to be dynamically generated ("Dynamic" branch of 116), then at 118 the content of the data block 2 is dynamically generated, preferably by a process external to the virtual filesystem 10, and at 122 the content is provided to the host computer in response to its request. If the data is to be aliased from another data memory such as an auxiliary file system 70 external to the VFS 10, then at 120 the content of the data block 2 is obtained from the auxiliary file system 70, and at 122 the content is provided to the host computer in response to its request..

[0033] By dynamically generating data blocks 2 as they are requested, or by obtaining data blocks 2 from an auxiliary file system 70 already present in an apparatus in which the virtual filesystem 10 is included, the present invention advantageously minimizes the amount of physical memory that is required by the virtual filesystem 10, particularly as large numbers of files and large file sizes are included in the filesystem 10.

[0034] The method 100 is preferably implemented in firmware and/or software as a program of instructions contained on a program storage medium such as a ROM, CD-ROM, floppy disk, and the like, executable by a computing apparatus (not shown) incorporated within the virtual filesystem 10. Such an implementation is well known to those skilled in the art. The program of instructions can be organized into a number of logical segments for performing the steps of the method 100 heretofore described to implement the file structure 15 and filespace 30 that have also been described.

[0035] Considering now in further detail the calculating 114 of the contents of a data block 2 in the FAT region 36 or directory region 32 based on the predetermined directory hierarchy 20, and as best understood with reference to FIGS. 2, 3, 4, and 6, at 130 the location of the requested data block within the FAT region 36 or directory region 32 of the filespace 30 is determined. If the requested data block is in the directory region 32 and corresponds to any directory other than a lowest-level (Yth level) directory, then at 132 the block number of the requested data block is converted to a directory-relative block number for the requested directory. At 134, the directory-relative block numbers of all child subdirectories of the requested directory are determined, preferably by calculating the block numbers of the child subdirectories based on the depth and fanout of the predetermined directory hierarchy 20 according to a conventional algorithm known to those skilled in the art for traversing an N-ary tree stored in an array. For example, if the requested block corresponds to directory-relative block #6, child subdirectories #25, #26, #27, and #28 are identified. Once the appropriate child subdirectories have been determined, then at 136, directory entry data for these child subdirectories of the requested directory are created.

[0036] If the requested data block is in the directory region 32 and corresponds to a lowest-level (Yth level) directory, then at 138 directory entry data representing a file is created. This data is derived from information contained in an entry 62 in the file shadow table 60 which corresponds to that lowest-level directory (the shadow table 60 and its entries 62 will be subsequently de-

scribed in further detail with reference to FIG. 7). For example, if the requested block corresponds to directory-relative block #48, then directory entry data 38d representing the file contained in the lowest-level directory associated with block #48 is created.

[0037] If the requested data block is in the FAT region 36, then at 140 the FAT entries contained in the requested data block are identified. In the preferred embodiment, these entries are readily calculated based on the size of the data block and the size of each FAT entry. Each FAT entry 42 refers to a data block 2 in the directory region 32 or the file region 34. At 142, it is determined which of these FAT entries correspond to blocks that are the last (or only) block in a block chain, and which correspond to blocks that are not the last (or only) block in a block chain. In the preferred embodiment, this can be readily calculated based on the organization of the filespace 30, since all directories occupy one data block, and the number of data files of each size M are predetermined. At 144, FAT entry data containing an end-of-chain marker is created for all FAT entries which correspond to the last (or the only) block in a block chain, while at 146 FAT entry data pointing to the next block in the chain is created for all FAT entries which do not correspond to the last (or the only) block in a block chain. For example, if the requested data block includes FAT entries for datatile-relative blocks #64 through #67, the FAT entry created for block #64 will point to block #65, while the FAT entry created for block #67 will contain the end-of-chain marker.

[0038] The present invention can also be embodied, as best understood with reference to FIG. 7, as an apparatus 50 which incorporates a virtual file system 10 to provide a block-oriented data source that can be accessed from a host computer (not shown). The VFS 10 further includes a region identifier 52 which receives "Read Block <B>" requests from the host computer, where <B> is the absolute data block number B in the VFS 10. The region identifier 52 determines whether block B corresponds to a block in the FAT region 36 or directory region 32 and can therefore be calculated as previously described, or whether block B corresponds to a block in the file region 34 and therefore file data must be obtained as previously described. In the preferred embodiment, the region identifier 52 converts the absolute block number B into a block number relative to the region 32,34,36 to which the block belongs. If the block is in the FAT region 36 or the directory region 32, a block calculator 54 which has knowledge of the predetermined file structure 15 along with the fanout N, depth Y, file quantums Q, and the number of blocks associated with each file quantum Q calculates the contents of the requested data block. If the block number corresponds to a Yth-level directory, a Yth-directory file binder 56 sends a request to a dynamic data generator 58 to bind a file shadow 62 in a file shadow table 60 to a particular predefined file of a given size in the filesystem 10.

[0039] The file shadow 62 has typically been previously created by an asynchronous request made to the dynamic data generator 58 by another module (not shown) of the apparatus 50. Examples of such requests are described in the above-referenced copending U.S. patent application to Holcomb et al. At the time the file shadow 62 is created, the amount of data to be associated with the shadow 62 may not be known, and therefore the file size (and in turn the file start block location in the file region 34) for the file cannot be known; the file shadow 62 thus typically contains only the Yth directory block number and a data source address or identifier at the time the shadow 62 is created. However, when the Yth directory is read, it is an advance notification that a subsequent request for the single file located in that Yth directory is in all probability imminent. In addition, in the preferred embodiment the size of the file must also be provided in the data block 2 returned for the Yth directory. Once the dynamic data generator 58 determines the file size, a particular predefined file that is at least as large as the file size can be selected, and therefore the file start block number and the file size fields in the shadow 62 are filled in and the particular file is bound to the Yth directory at that time.

[0040] In order to eliminate or minimize any possibility that the host computer may think it already contains a particular data block in its cache memory (not shown) and thus not request it from the apparatus 50, the apparatus 50 preferably does two things. First, the dynamic data generator 58, when processing a request to create a file shadow 62, establishes a pathname that has never been used before (by choosing a Yth directory block that has never been used before. And second, when processing a request to bind a file shadow 62 to a file, the dynamic data generator 58 selects a file that has never been used before. Because the physical memory size of the virtual filesystem 10 is largely insensitive to the number and sizes of the files that the filesystem 10 contains, the virtual filesystem 10 for a particular apparatus 50 can be chosen so as to minimize any chance that a Yth-level directory or a file need ever be used twice. However, the preferred embodiment of the apparatus also includes a block manager 64 which assigns Yth-level directories and files upon request. The block manager 64 keeps track of the usage of Yth-level directories and files by receiving notification from the Yth-directory file binder 56 and the file generator 66 as blocks are requested by the host computer. If the apparatus 50 runs out of unused directories or files, the block manager 64 ensures that a least recently used (LRU) one is assigned in the expectation that any previous information about that directory or file has already expired from the host computer's cache. The algorithms used by the block manager 64 are similar to those used in memory management and garbage collection, and are well known to those skilled in the art.

[0041] If the requested block B is in the file region 34, a file generator 66 searches the file shadow table 60 to

locate the file shadow 62 associated with the block. The data source field of the file shadow 62 is then used by the file generator 66 to determine the source of the data. If the source is the dynamic data generator 58, the block is requested from the dynamic data generator 58. If the source is a file on an auxiliary file system 70, a file aliasing bridge 68 translates the requested block into the corresponding block of the file on the auxiliary file system 70. One example of the auxiliary file system 70 is the file system utilized by a Compact Flash card or PCMCIA flash card which is used, for example, to store photographic image files recorded by a digital camera.

[0042] From the foregoing it will be appreciated that the virtual filesystem provided by the present invention represents a significant advance in the art. Although several specific embodiments of the invention have been described and illustrated, the invention is not limited to the specific methods, forms, or arrangements of parts so described and illustrated. In particular, the filesystem may contain additional regions such as boot sectors, and may contain a redundant copy of the FAT region. It is also to be understood that the root directory and intermediate-level directories may contain additional predefined directories or files that have been omitted for clarity, including but not limited to a self-referencing directory (usually denoted as the "." directory) and a parent directory (usually denoted as the ".." directory). Furthermore, while in the preferred embodiment the file size is fixed when a Yth-level directory is accessed, in some embodiments the binding may be deferred until the file itself is accessed, depending on the behavior of the host computer's operating system. The present invention may be advantageously used in a wide variety of traditional and embedded computer systems. The invention is limited only by the claims.

Claims

1. A method (100) for reading data from a virtual filesystem (10), comprising a virtual file system (10), comprising:

   a filespace (30) including a plurality of logical data blocks (2), comprising:

   a first contiguous region (32) in the filespace (30) predefining a directory hierarchy (20) of the filesystem (10), the hierarchy (20) predefining an interconnected hierarchy of directories;
   a plurality of second contiguous regions (34) in the filespace (30) each predefining a set of data files (18) having a predetermined number of data blocks (2); and
   a third contiguous region (36) in the filespace (30) predefining a file allocation table for locating in the filespace (30) all the data blocks (2) for the directories (20) and the data files (18);
   the regions organized so as to allow the structure of the data blocks (2) in the first and third regions (32, 36) to be algorithmically calculated when the filesystem (10) is read in order to reduce the amount of physical memory required to contain the filesystem (10);
   the method comprising the steps of:

   receiving a block read request for a specified data block (2);
   determining whether the specified data block (2) corresponds to the directory hierarchy (20) or the set of data files (18);
   algorithmically calculating data for the specified block (2) when the block (2) corresponds to the directory hierarchy (20); and
   providing the data for the specified block (2) when the block (2) corresponds to the set of data files (18).

2. The method of claim 1, wherein the predetermined hierarchy of directories (20) further includes a set of lowest-level directories, each lowest-level directory having a predetermined file placeholder, wherein the data block (2) not stored in a physical memory corresponds to a selected one of the lowest-level directories, and wherein the file structure (15) includes a predetermined set of files comprising one or several data blocks provided in the virtual file system, further comprising:

   binding (112) the file placeholder for the selected one of the lowest-level directories to a corresponding one of the predetermined set of files provided in the virtual file system.

3. The method of claim 2, wherein each of the predetermined set of files comprising one or several data blocks has a maximum file size, further comprising:

   determining (110) an amount of data associated with the file placeholder, and
   choosing (112) the corresponding one of the predetermined set of files so as to have a maximum file size at least equal to the amount of data.

4. The method of claim 2 or 3, further comprising:

   receiving (104) an additional request to provide the content of an additional non-stored data block (2) associated with the corresponding one of the predetermined set of files; and

generating the content of the additional non-stored data block (2).

5. The method of claim 4, wherein the generating includes:

dynamically generating (118) the content in response to the additional request by a process external to the virtual file system.

6. The method of claim 4 or 5, wherein the generating includes:

obtaining (120) the content from an auxiliary file system (70) external to the virtual file system.

7. The method of one of the preceding claims, wherein the virtual file structure (15) provides a linear logical arrangement (30) of the data blocks (2) including a directory hierarchy and a set of placeholders for predetermined files (18).

8. The method of claim 3, wherein the binding (112) further comprises:

determining (110) the actual file size;
locating a least recently used file placeholder at least equal to the actual file size; and
assigning the least recently used file placeholder as the selected one of the predetermined file placeholders.

9. An apparatus (50) for providing a virtual file system (10), comprising:

a filespace (30) including a plurality of logical data blocks (2), comprising:

a first contiguous region (32) in the filespace (30) predefining a directory hierarchy (20) of the filesystem (10), the hierarchy (20) predefining an interconnected hierarchy of directories;
a plurality of second contiguous regions (34) in the filespace (30) each predefining a set of data files (18) having a predetermined number of data blocks (2); and
a third contiguous region (36) in the filespace (30) predefining a file allocation table for locating in the filespace (30) all the data blocks (2) for the directories (20) and the data files (18);
the regions organized so as to allow the structure of the data blocks (2) in the first and third regions (32, 36) to be algorithmically calculated when the filesystem (10) is read in order to reduce the amount of physical memory required to contain the filesystem (10);
a region identifier (52) adapted to receive a block read request for a specified data block (2) and determine whether the specified data block (2) corresponds to the directory hierarchy (20) or the set of data files (18);
a block calculator (54) connected to the region identifier (52) to algorithmically calculate data for the specified block (2) when the block (2) corresponds to the directory hierarchy (20); and
a file generator (68) connected to the region identifier (52) to provide the data for the specified block (2) when the block (2) corresponds to the set of data files (18);

10. The apparatus of claim 9, wherein the directories comprise a root-level directory, a plurality of intermediate-level directories below the root-level directory, and a plurality of lowest-level directories below the intermediate-level directories.

11. The apparatus of claim 9 or 10, wherein the block calculator (54) further detects whether the specified block (2) corresponds to any of a set of lowest-level directories, the apparatus (50) further comprising:

a dynamic data generator (58) for establishing a file shadow (62) corresponding to a selected one of the lowest-level directories and determining a file size for the data; and
a file binder (56) connected to the block calculator (54) and the dynamic data generator (58) for determining whether the specified block (2) corresponds to the file shadow (62) and, if so, obtaining the file size from the dynamic data generator (58) and binding the selected one of the lowest-level directories to an appropriately-sized one of the set of data files (18) at least as large as the file size.

12. The apparatus of one of claims 9 to 11, wherein a selected one of the set of data files (18) is linked to a specified one of the lowest-level directories when the specified one is first accessed.

Patentansprüche

1. Verfahren (100) zum Lesen aus einem virtuellen Dateisystems (10), mit einem virtuellen Dateisystem (10), umfassend:

einem Dateibereich (30) mit einer Vielzahl logischer Datenblöcke (2), mit:

einen ersten zusammenhängenden Be-

reich (32) in dem Dateibereich (30), welcher eine Verzeichnishierarchie (20) des Dateisystems (10) vordefiniert, wobei die Hierarchie (20) eine in sich verknüpfte Hierarchie von Verzeichnissen vordefiniert; einer Vielzahl zweiter zusammenhängender Bereiche (34) in dem Dateibereich (30), die jeweils einen Satz Datendateien (18) mit einer vorbestimmten Anzahl von Datenblökken (2) vordefinieren; und einem dritten zusammenhängenden Bereich (36) in dem Dateibereich (30), der eine Dateizuordnungstabelle vordefiniert, um in dem Dateibereich (30) alle diese Datenblöcke (2) hinsichtlich der Verzeichnisse (20) und der Datendateien (18) anzuordnen;

wobei die Bereiche so organisiert sind, daß die Struktur der Datenblöcke (2) in dem ersten und dem dritten Bereich (32, 36) algorithmisch berechnet werden kann, wenn das Dateisystem (10) gelesen wird, um die Größe des physikalischen Speichers zu reduzieren, der zur Aufnahme des Dateisystems (10) notwendig ist,
wobei das Verfahren die Schritte umfaßt:

Empfangen einer Blocklese-Anfrage bezüglich eines spezifizierten Datenblocks (2); Ermitteln, ob der spezifizierte Datenblock (2) der Verzeichnishierarchie (20) oder dem Satz Datendateien (18) entspricht;
algorithmisches Berechnen von Daten für den bestimmten Block (2), wenn der Block (2) der Verzeichnishierarchie (20) entspricht; und
Vorsehen der Daten für den bestimmten Block (2), wenn der Block (2) dem Satz Datendateien (18) entspricht.

2. Verfahren nach Anspruch 1, wobei die vorbestimmte Hierarchie von Verzeichnissen (20) ferner einen Satz Verzeichnisse der untersten Ebene umfaßt, wobei jedes Verzeichnis der untersten Ebene einen vorbestimmten Datei-Platzhalter aufweist, wobei der nicht in einem physikalischen Speicher gespeicherte Datenblock (2) einem aus den Verzeichnissen der untersten Ebene ausgewählten Verzeichnis entspricht, und die Dateistruktur (15) einen vorbestimmten Satz von Dateien umfaßt, der einen oder mehrere Datenblöcke umfaßt, die in dem virtuellen Dateisystem vorgesehen sind, und das Verfahren ferner umfaßt:

Binden (112) des Datei-Platzhalters, der für das ausgewählte Verzeichnis der Verzeichnisse der untersten Ebene steht, an eine entsprechende Datei des vorbestimmten Satzes von Dateien, die in dem virtuellen Dateisystem vorgesehen sind,

3. Verfahren nach Anspruch 2, wobei jede Datei des vorbestimmten Satzes von Dateien, der ein oder mehrere Datenblöcke umfaßt, eine maximale Dateigröße aufweist, wobei das Verfahren ferner umfaßt:

Ermitteln (110) einer Datenmenge, die mit dem Dateiplatzhalter verknüpft ist, und Auswählen (112) der entsprechenden Datei aus dem vorbestimmten Satz von Dateien, so daß dieser eine maximale Dateigröße mindestens gleich der Datenmenge aufweist.

4. Verfahren nach Anspruch 2 oder 3, daß ferner umfaßt:

Empfangen (104) einer weiteren Anfrage, den Inhalt eines weiteren, nicht gespeicherten Datenblocks (2) vorzusehen, der mit der entsprechenden Datei des vorbestimmten Satzes von Dateien verknüpft ist; und
Erzeugen des Inhalts des zusätzlichen, nicht gespeicherten Datenblocks (2).

5. Verfahren nach Anspruch 4, wobei das Erzeugen umfaßt:

Dynamisches Erzeugen (118) des Inhalts als Reaktion auf die zusätzliche Anfrage mittels eines Prozesses außerhalb des virtuellen Dateisystems.

6. Verfahren nach Anspruch 4 oder 5, wobei das Erzeugen umfaßt:

Erhalten (120) des Inhalts aus einem Neben-Dateisystem (70) außerhalb des virtuellen Dateisystems.

7. Verfahren nach einem der vorangegangenen Ansprüche, wobei die virtuelle Dateistruktur (15) eine lineare logische Anordnung (30) der Datenblöcke (2) umfaßt, einschließlich einer Verzeichnishierarchie und eines Satzes von Platzhaltern für vorbestimmte Dateien (18).

8. Verfahren nach Anspruch 3, wobei das Binden (112) ferner umfaßt:

Ermitteln (110) der aktuellen Dateigröße;
Lokalisieren von zumindest einem zuletzt verwendeten Dateiplatzhalters, der mindestens so groß wie die aktuellen Dateigröße ist; und
Bestimmen des am längsten unbenutzten Dateiplatzhalters als den ausgewählten Platzhalter der vorbestimmten Dateiplatzhalter.

9. Vorrichtung (50) zum Vorsehen eines virtuellen Dateisystems (10), umfassend:

einen Dateibereich (30) mit einer Vielzahl logischer Datenblöcke (2), umfassend:

einen ersten zusammenhängenden Bereich (32) in dem Dateibereich (30), welcher eine Verzeichnishierarchie (20) des Dateisystems (10) vordefiniert, wobei die Hierarchie (20) eine in sich verknüpfte Hierarchie von Verzeichnissen vordefiniert;
eine Vielzahl zweiter zusammenhängender Bereiche (34) in dem Dateibereich (30), die jeweils einen Satz Datendateien (18) mit einer vorbestimmten Anzahl von Datenblöcken (2) vordefinieren; und
einen dritten zusammenhängenden Bereich (36) in dem Dateibereich (30), der eine Dateizuordnungstabelle vordefiniert, um in dem Dateibereich (30) alle diese Datumsblöcke (2) hinsichtlich der Verzeichnisse (20) und der Datendateien (18) anzuordnen;

wobei die Bereiche so organisiert sind, daß die Struktur der Datenblöcke (2) in dem ersten und dem dritten Bereich (32, 36) algorithmisch berechnet werden kann, wenn das Dateisystem (10) gelesen wird, um die Größe des physikalischen Speichers zu reduzieren, der zur Aufnahme des Dateisystems (10) notwendig ist;
einen Bereichsbezeichner (52), der vorgesehen ist, eine Blocklese-Anfrage bezüglich eines bestimmten Datenblocks (2) zu empfangen und zu ermitteln, ob der bestimmte Datenblock (2) der Verzeichnishierarchie (20) oder dem Satz von Datendateien (18) entspricht;
einen Blockberechner (54), der mit dem Bereichsbezeichner (52) verbunden ist, um Daten für den spezifizierten Block algorithmisch zu berechnen, wenn der Block (2), der Verzeichnishierarchie (20) entspricht; und
einen Dateigenerator (66), der mit dem Bereichsbezeichner (52) verbunden ist, um die Daten für den bestimmten Block (2) vorzusehen, wenn der Block (2) dem Satz von Datendateien (18) entspricht.

10. Vorrichtung nach Anspruch 9, wobei die Verzeichnisse ein Root-Ebene-Verzeichnis, eine Vielzahl von Zwischenebene-Verzeichnissen unterhalb des Root-Ebene-Verzeichnisses sowie eine Vielzahl von Verzeichnissen der untersten Ebene unterhalb der Zwischenebene-Verzeichnisse umfaßt.

11. Vorrichtung nach Anspruch 9 oder 10, wobei der Blockberechner (54) ferner ermittelt, ob der bestimmte Block (2) einem Verzeichnis eines Satzes von Verzeichnissen der untersten Ebene entspricht, wobei die Vorrichtung (50) ferner umfaßt:

einen dynamischen Datengenerator (58) zum Einrichten einer Dateispiegelung (62), die einem ausgewählten Verzeichnis der Verzeichnisse der untersten Ebene entspricht und eine Dateigröße für die Daten ermittelt; und
einen Dateibinder (56), der mit dem Blockberechner (54) und dem dynamischen Datengenerator (58) verbunden ist um zu ermitteln, ob der bestimmte Block (2) der Dateispiegelung (62) entspricht, und für diesen Fall die Dateigröße von dem dynamischen Datengenerator (58) erhält und das ausgewählte Verzeichnis der Verzeichnisse der untersten Ebene an eine geeignet bemessene Datendatei des Satzes von Datendateien (18) bindet, die mindestens so groß wie die Dateigröße ist.

12. Vorrichtung nach einem der Ansprüche 9 bis 11, wobei eine ausgewählte Datendatei des Satzes von Datendateien (18) mit einem bestimmten Verzeichnis der untersten Ebene verknüpft ist, wenn auf die bestimmte Datendatei zuerst zugegriffen wird.

Revendications

1. Un procédé (100) de lecture de données dans un système virtuel (10) de fichiers, comprenant un système virtuel (10) de fichiers, comprenant :

un espace (30) de fichiers incluant une série de blocs de données logiques (2) comprenant :

une première région contiguë (32) contenue dans l'espace (30) de fichiers et prédéfinissant une hiérarchie (20) de répertoires du système (10) de fichiers, la hiérarchie prédéfinissant une hiérarchie interconnectée de répertoires (20) ;
une série de deuxièmes régions contiguës (34) contenues dans l'espace (30) de fichiers et prédéfinissant chacune un jeu de fichiers (18) de données incluant un nombre prédéterminé de blocs (2) de données;
une troisième région contiguë (36) contenue dans l'espace (30) de fichiers et prédéfinissant une table d'allocation de fichiers pour localiser dans l'espace (30) de fichiers tous les blocs (2) de données pour les répertoires (20) et les fichiers (18) de données;
les régions étant organisées de manière à permettre à la structure des blocs (2) de données des première et troisième régions

(32, 36) d'être calculée de façon algorithmique lorsque le système (10) de fichiers est lu afin de réduire la quantité de mémoire physique requise pour contenir le système (10) de fichiers;
le procédé comprenant les étapes consistant à:

recevoir une demande de lecture de bloc pour un bloc spécifié (2) de données;
déterminer si le bloc spécifié (2) de données correspond à la hiérarchie (20) de répertoires ou au jeu des fichiers (18) de données;
calculer de façon algorithmique des données pour le bloc spécifié (2) lorsque le bloc (2) correspond à la hiérarchie (20) de répertoire; et
envoyer les données pour le bloc spécifié (2) lorsque le bloc (2) correspond au jeu de fichiers (18) de données.

2. Le procédé selon la revendication 1, dans lequel la hiérarchie prédéterminé de répertoires (20) inclut en outre un jeu de répertoires du niveau le plus bas, chaque répertoire du niveau le plus bas incluant un emplacement réservé prédéterminé de fichier, dans lequel le bloc (2) de données qui n'est pas enregistré dans une mémoire physique correspond à un répertoire sélectionné parmi les répertoires du niveau le plus bas, et dans lequel la structure (15) de fichiers inclut un jeu prédéterminé de fichiers comprenant un ou plusieurs blocs de données contenus dans le système virtuel de fichiers, qui comprend en outre l'étape consistant à:

lier (112) l'emplacement réservé de fichier pour le répertoire sélectionné parmi les répertoires du niveau le plus bas à un fichier correspondant du jeu prédéterminé de fichiers contenu dans le système virtuel de fichiers.

3. Le procédé selon la revendication 2, dans lequel chaque fichier du jeu prédéterminé de fichiers comprenant un ou plusieurs blocs de données est d'une dimension maximale de fichiers, le procédé comprenant en outre les étapes consistant à:

déterminer (110) une quantité de données associées à l'emplacement réservé de fichier, et choisir le fichier correspondant du jeu prédéterminé de fichiers d'une manière telle que la dimension maximale du fichier soit au moins égale à la quantité de données.

4. Le procédé selon la revendication 2 ou 3, qui comprend en outre les étapes consistant :

recevoir (104) une demande additionnelle d'envoi du contenu d'un bloc additionnel non enregistré (2) de données associé au fichier correspondant du jeu prédéterminé de fichiers; et
engendrer le contenu du bloc non enregistré additionnel (2) de données.

5. Le procédé selon la revendication 4, dans lequel la génération inclut l'étape consistant à:

engendrer dynamiquement (118) le contenu en réponse à la demande additionnelle par un processus externe au système virtuel de fichiers.

6. Le procédé selon la revendication 4 ou 5, dans lequel la génération inclut l'étape consistant à:

obtenir (120) le contenu à partir d'un système auxiliaire (70) de fichiers, externe au système virtuel de fichiers.

7. Le procédé selon l'une quelconque des revendications précédentes, dans lequel la structure virtuelle (15) de fichiers réalise un agencement logique linéaire (30) des blocs (2) de données incluant une hiérarchie de répertoires et un jeu d'emplacements réservés pour des fichiers prédéterminés (18).

8. Le procédé selon la revendication 3, dans lequel l'édition (112) de liens comprend en outre les étapes consistant à:

déterminer (110) la dimension réelle du fichier;
localiser un emplacement réservé utilisé le moins récemment de fichier au moins égal à la dimension réelle de fichier; et
assigner l'emplacement réservé utilisé le moins récemment de fichier comme l'emplacement réservé sélectionné parmi les emplacements réservés prédéterminés de fichiers.

9. Un appareil (50) de réalisation d'un système virtuel (10) de fichiers comprenant:

un espace (30) de fichiers incluant une série de blocs de données logiques (2) comprenant:

une première région contiguë (32) contenue dans l'espace (30) de fichiers et prédéfinissant une hiérarchie (20) de répertoires du système (10) de fichiers, la hiérarchie (20) prédéfinissant une hiérarchie interconnectée de répertoires;
une série de deuxièmes régions contiguës (34) contenues dans l'espace (30) de fichiers et prédéfinissant chacune un jeu de fichiers (18) de données incluant un nombre prédéterminé de blocs (2) de données;

une troisième région contiguë (36) contenue dans l'espace (30) de fichiers et prédéfinissant une table d'allocation de fichiers pour localiser dans l'espace (30) de fichiers tous les blocs (2) de données pour les répertoires (20) et les fichiers (18) de données;
les régions étant organisées de manière à permettre à la structure des blocs (2) de données des première et troisième régions (32, 36) d'être calculée de façon algorithmique lorsque le système (10) de fichiers est lu afin de réduire la quantité de mémoire physique requise pour contenir le système (10) de fichiers;
un identifiant (52) de région apte à recevoir une demande de lecture de bloc pour un bloc spécifié (2) de données et à déterminer si le bloc spécifié (2) de données correspond à la hiérarchie (20) de répertoires ou au jeu de fichiers (18) de données;
un calculateur (54) de blocs connecté à l'identifiant (52) de région pour calculer de façon algorithmique des données pour le bloc spécifié (2) lorsque le bloc (2) correspond à la hiérarchie (20) de répertoire; et
un générateur (66) de fichiers connecté à l'identifiant (52) de région pour envoyer les données pour le bloc spécifié (2) lorsque le bloc (2) correspond au jeu de fichiers (18) de données.

10. L'appareil selon la revendication 9, dans lequel les répertoires comprennent un répertoire du niveau racine, une série de répertoires de niveaux intermédiaires au-dessous du répertoire de niveau racine, et une série de répertoires du niveau le plus bas au-dessous des répertoires de niveaux intermédiaires.

11. L'appareil selon la revendication 9 ou 10, dans lequel le calculateur (54) de blocs détecte en outre si le bloc spécifié (2) correspond à un répertoire quelconque d'un jeu de répertoires du niveau le plus bas, l'appareil (50) comprenant en outre:

un générateur dynamique (58) de données pour établir un fantôme (62) de fichier correspondant à un répertoire sélectionné parmi les répertoires du niveau le plus bas et pour déterminer une dimension de fichier pour les données; et
un éditeur (56) de liens de fichier connecté au calculateur (54) de bloc et au générateur dynamique (58) de données pour déterminer si le bloc spécifié (2) correspond au fantôme (62) de fichier et, si tel est le cas, obtenir du générateur dynamique (58) de données la dimension de fichier et lier le répertoire sélectionné parmi les répertoires du niveau le plus bas à un fichier de dimension appropriée du jeu de fichiers (18) de données, au moins aussi grand que la dimension du fichier.

12. L'appareil selon l'une quelconque des revendications 9 à 11, dans lequel un fichier sélectionné du jeu de fichiers (18) de données est lié à un répertoire spécifié parmi les répertoires du niveau le plus bas lors du premier accès à ce répertoire spécifié.

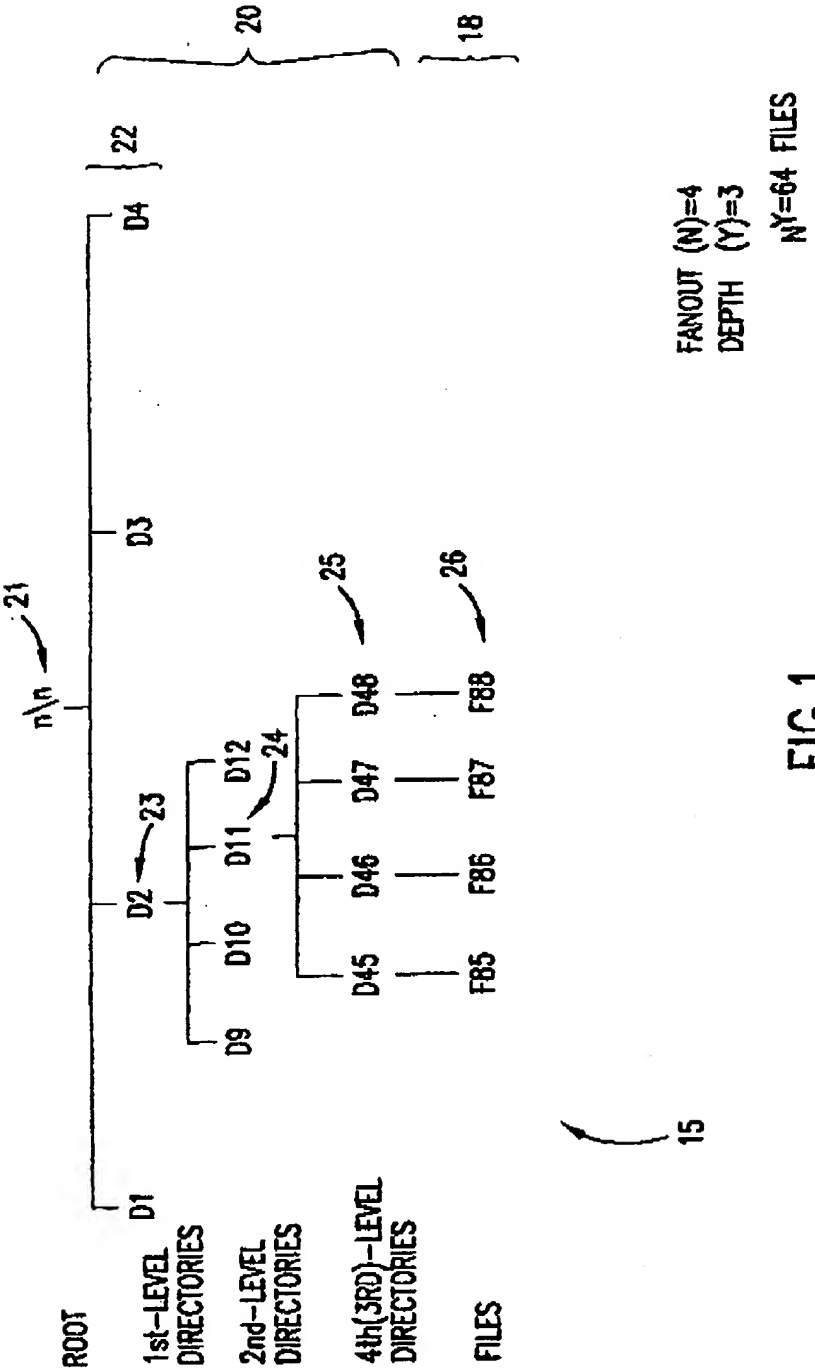
20
18
22
D4
D3
21
n\n
25
26
D48
F88
D12
24
D47
F87
23
D2
D11
D46
F86
D10
D45
F85
D9
15
D1
ROOT
1st-LEVEL DIRECTORIES
2nd-LEVEL DIRECTORIES
4th(3RD)-LEVEL DIRECTORIES
FILES
FANOUT (N)=4
DEPTH (Y)=3
N^Y=64 FILES

FIG.1

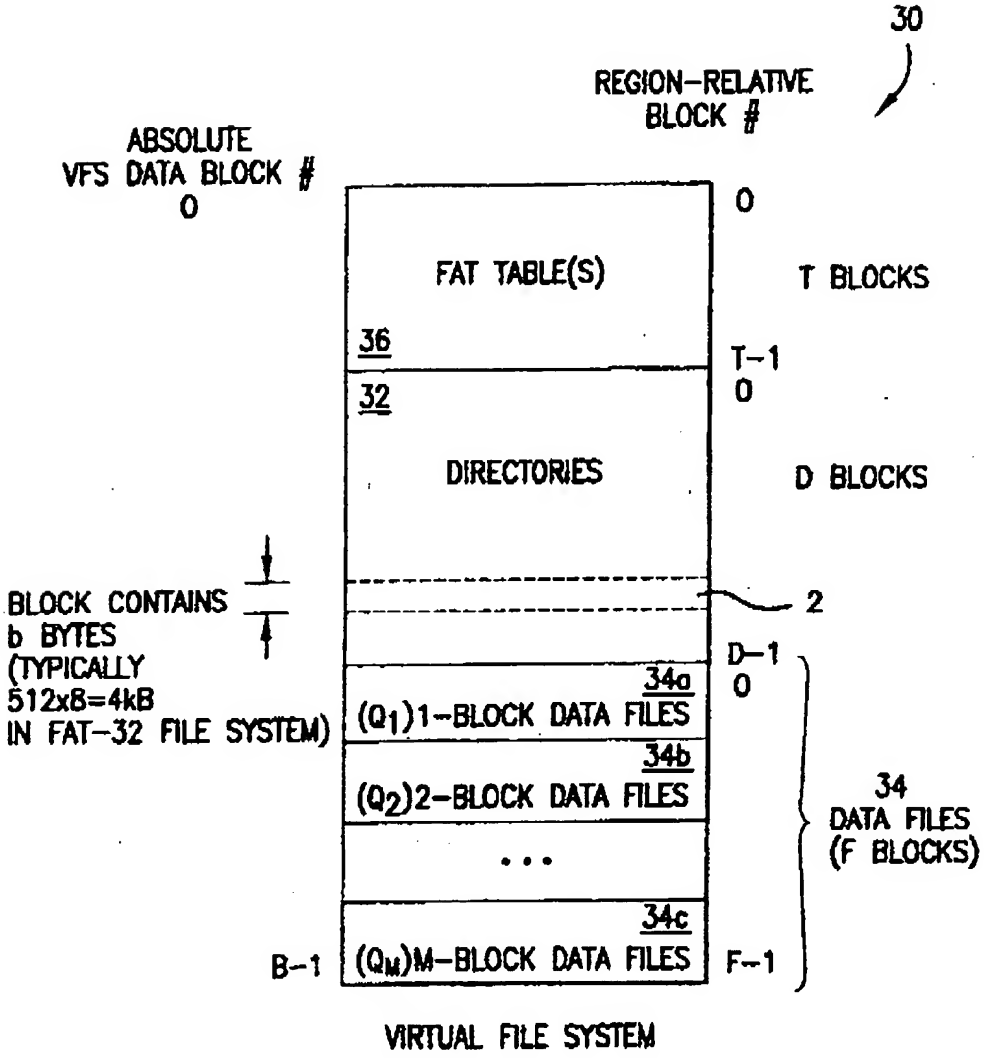

FIG.2

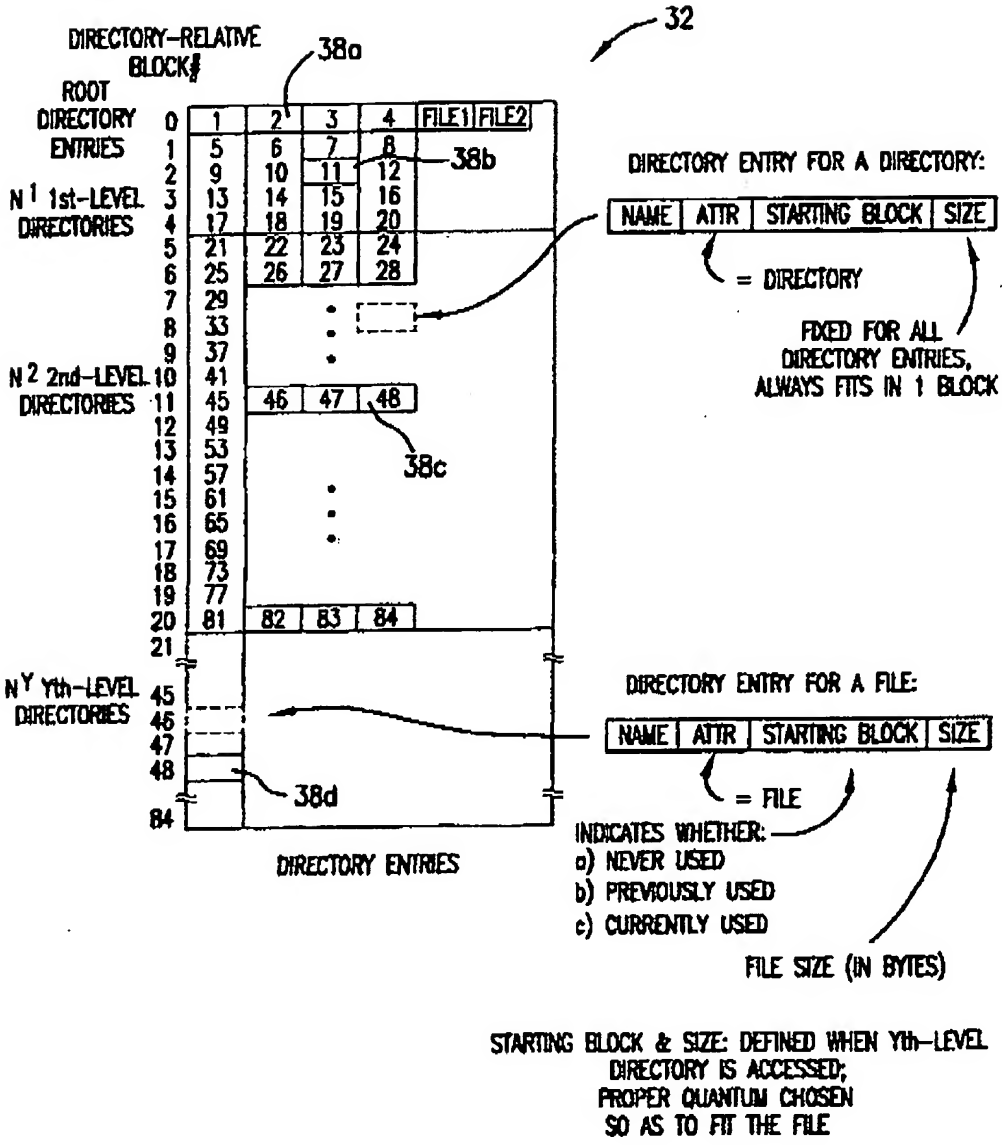

FIG.3

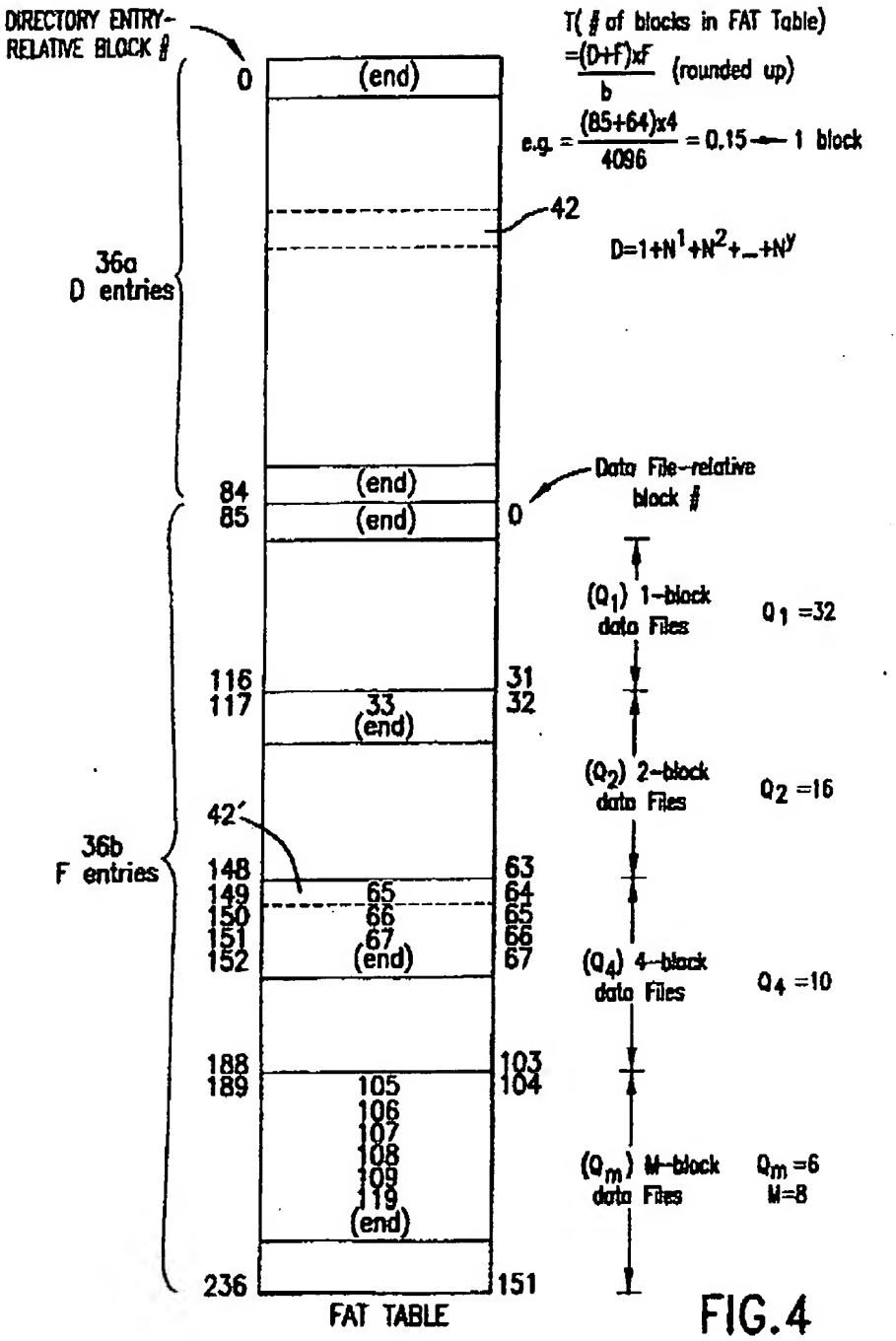

FIG.4

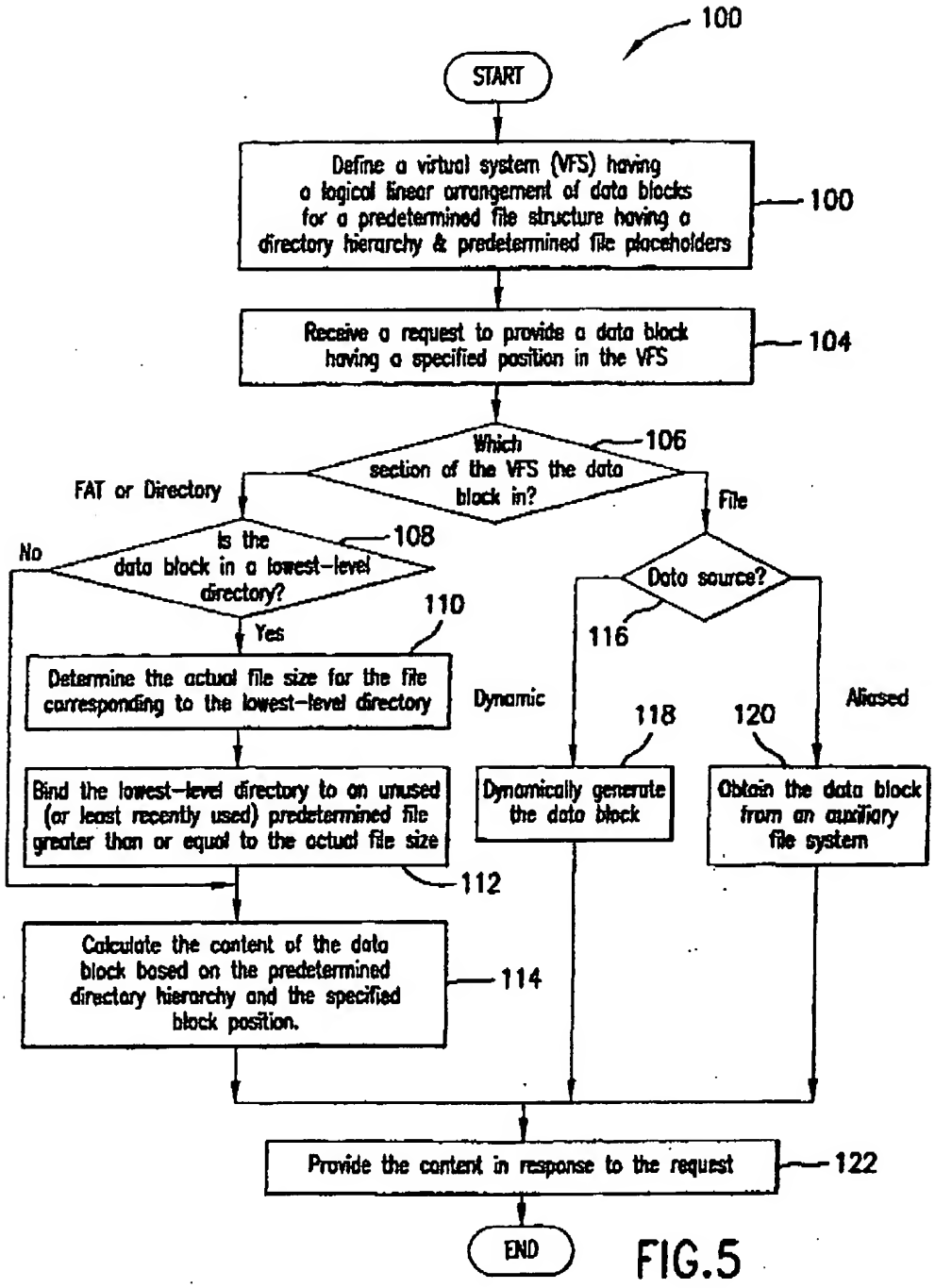
100
START
Define a virtual system (VFS) having a logical linear arrangement of data blocks for a predetermined file structure having a directory hierarchy & predetermined file placeholders
100
Receive a request to provide a data block having a specified position in the VFS
104
Which section of the VFS the data block in?
106
FAT or Directory
File
No
Is the data block in a lowest-level directory?
108
Yes
110
Determine the actual file size for the file corresponding to the lowest-level directory
Bind the lowest-level directory to an unused (or least recently used) predetermined file greater than or equal to the actual file size
112
Data source?
116
Dynamic
Aliased
118
120
Dynamically generate the data block
Obtain the data block from an auxiliary file system
Calculate the content of the data block based on the predetermined directory hierarchy and the specified block position.
114
Provide the content in response to the request
122
END

FIG.5

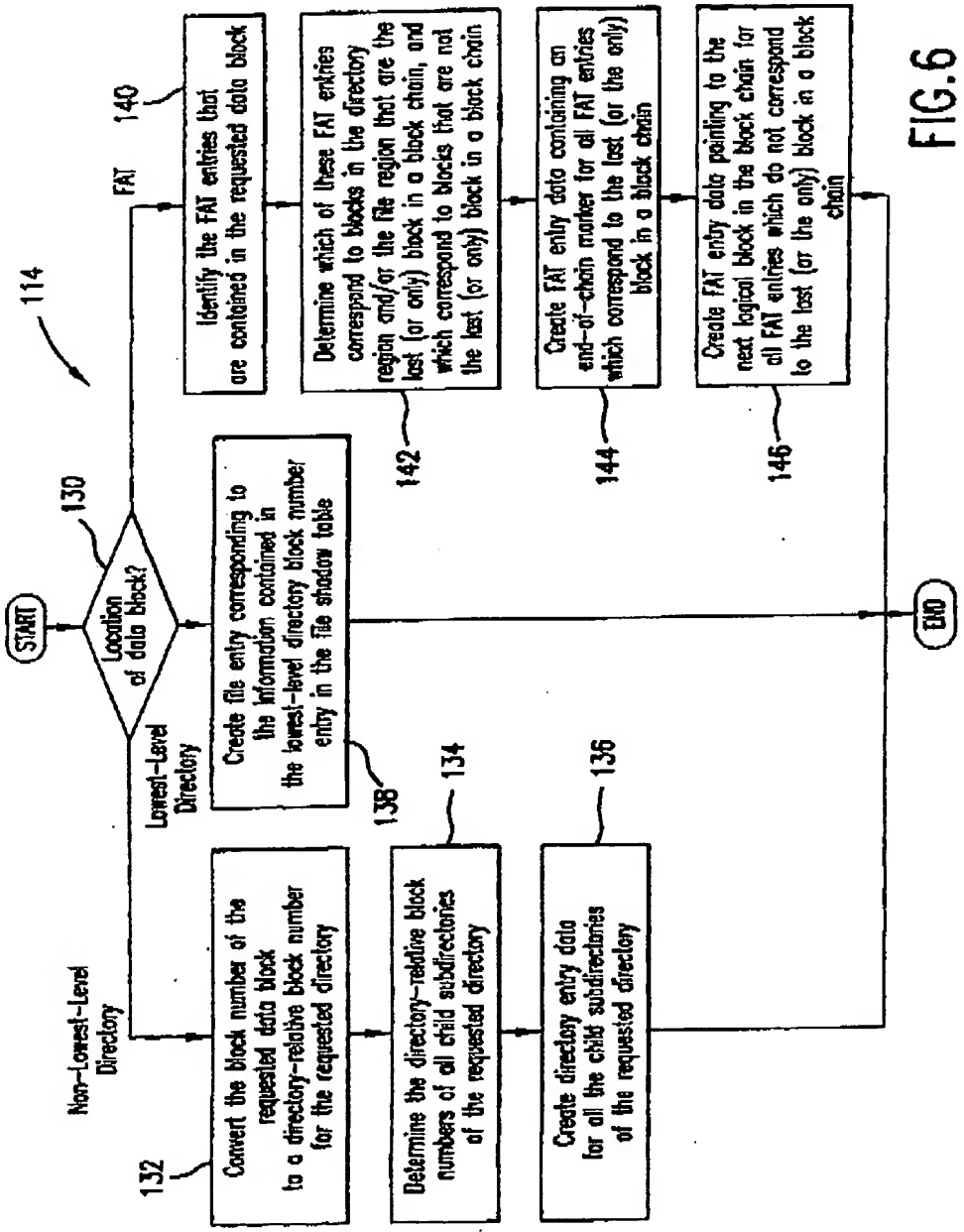
START
Location of data block?
114
130
FAT
140
Identify the FAT entries that are contained in the requested data block
Determine which of these FAT entries correspond to blocks in the directory region and/or the file region that are the last (or only) block in a block chain, and which correspond to blocks that are not the last (or only) block in a block chain
142
Create FAT entry data containing an end-of-chain marker for all FAT entries which correspond to the last (or the only) block in a block chain
144
Create FAT entry data pointing to the next logical block in the block chain for all FAT entries which do not correspond to the last (or the only) block in a block chain
146
Lowest-Level Directory
Create file entry corresponding to the information contained in the lowest-level directory block number entry in the file shadow table
138
END
Non-Lowest-Level Directory
132
Convert the block number of the requested data block to a directory-relative block number for the requested directory
Determine the directory-relative block numbers of all child subdirectories of the requested directory
134
Create directory entry data for all the child subdirectories of the requested directory
136

FIG.6

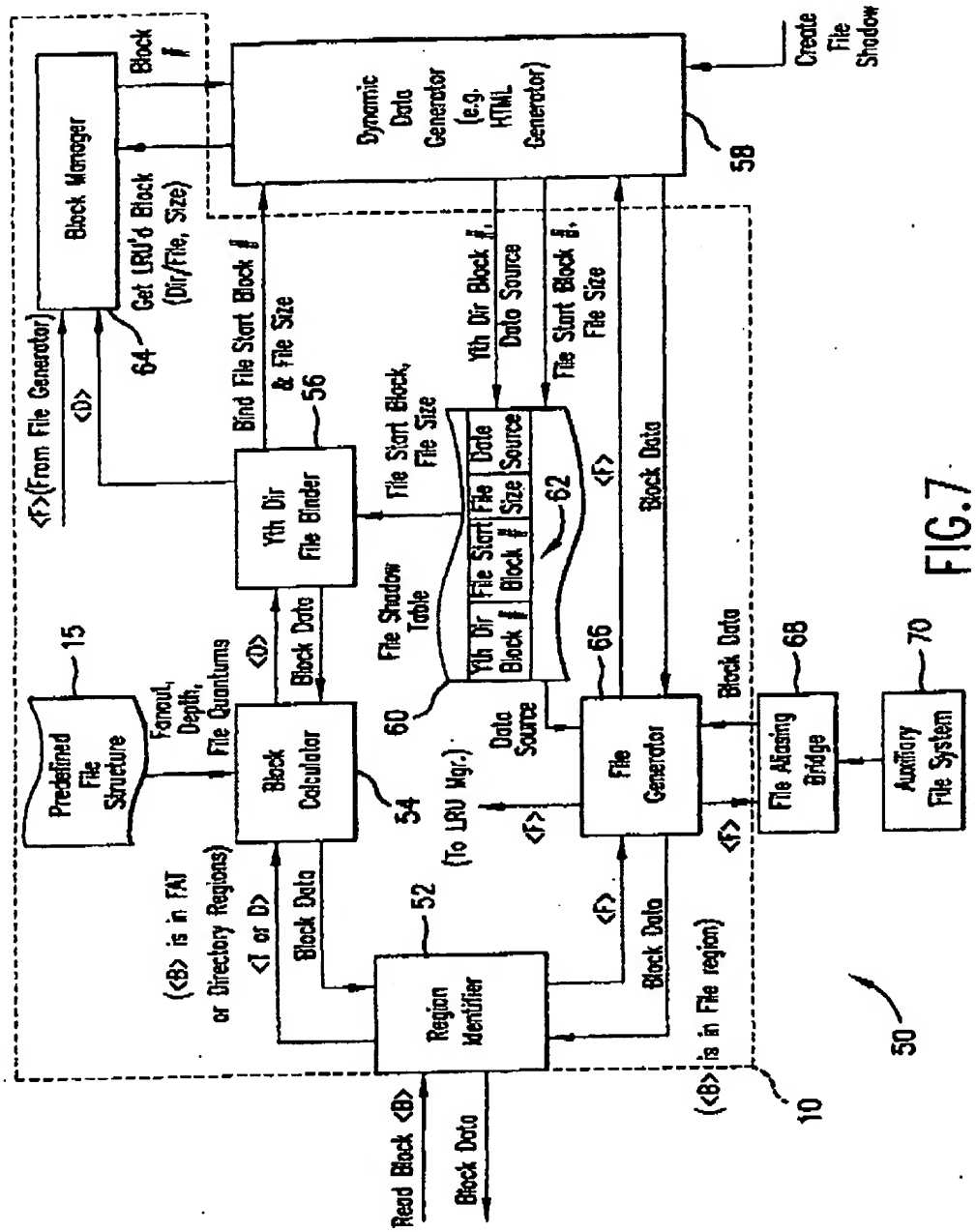
Block Manager
Block #
Get LRU'd Block (Dir/File, Size)
64
<F>(From File Generator)
<D>
Dynamic Data Generator (e.g. HTML Generator)
Create File Shadow
58
Bind File Start Block # & File Size
Yth Dir File Binder
56
File Start Block, File Size
Yth Dir Block #, Data Source
File Start Block #, File Size
File Shadow Table
Yth Dir Block # | File Start Block # | File Size | Data Source
62
60
<F>
Block Data
<D>
Block Data
15
Predefined File Structure
Fanout, Depth, File Quantums
Block Calculator
54
Data Source
(To LRU Mgr.)
File Generator
66
Block Data
File Aliasing Bridge
68
Auxiliary File System
70
<F>
<F>
(<B> is in FAT or Directory Regions)
<T or D>
Block Data
52
Region Identifier
<F>
Block Data
(<B> is in File region)
50
10
Read Block <B>
Block Data

FIG.7